Fast Walsh–Hadamard Transform in Neural Networks (Clean View)

1. Start With the Right Mental Model

Think of the Fast Walsh–Hadamard Transform as:

> A *cheap, dense, orthogonal mixing layer* that turns point-wise data into global patterns—and back again.

It's not doing anything "mystical"—it's just:

- a matrix with ±1 entries
- orthogonal rows
- fast to compute O(nlog(n))

But what matters is **what it does to information flow**.

---

2. Case 1: Raw Input Data (Where Spectral Bias Matters)

Real-world inputs (like images, audio, etc.) are:

- structured
- correlated
- not "uniformly distributed" across basis directions

If you apply FWHT directly:

- energy tends to concentrate in certain components
- you don't get a clean "spread" of information

Fix: Random Sign Flips First

Before applying FWHT:

$$x' = Dx$$

- D: diagonal matrix of random ±1 (or low-discrepancy / sub-random pattern)

Then:

$\hat{x} = Hx'$

Why this works

- Breaks alignment with the Walsh basis
- Prevents systematic concentration of energy
- Produces a more **uniform / fair distribution across components**

Intuition

> Without sign flips: transform “sees structure”
> With sign flips: transform “sees randomness” → spreads information evenly

---

3. Case 2: Inside the Network (Where Bias Doesn’t Matter)

Once you’re inside the network, things change.

At this point:

- inputs to each layer are already **learned mixtures**
- they’re not raw, structured signals anymore

So the FWHT inside the network can be treated as:

> Just a dense orthogonal matrix—no special precautions needed

Important shift in perspective

Don’t think about:

- “pixels”
- “input coordinates”

Instead think about:

- **layer outputs as abstract feature vectors**

---

4. What FWHT Does Between Layers

Each time you apply FWHT:

$x^{(l)} \rightarrow Hx^{(l)}$

you get:

- every feature mixed with every other feature
- local information becomes **global immediately**

So the pipeline becomes:

(layer output)→H→(next layer)

---

5. Why This Enables Shallow Depth

Normally:

- networks need depth to gradually mix information

Here:

- mixing happens **in one step**

So even if your layers are:

- small
- sparse
- local

they still operate on:

globally mixed representations

Result

- A shallow stack of layers can behave like a deep network
- Because each layer already has access to the *entire state*

---

6. Local Computation Becomes Global

A neuron that looks simple:

- before FWHT:
  - depends on a few coordinates
- after FWHT:
  - depends on **patterns over all coordinates**

So:

> Small local operations become **global conditional operations**

---

7. Layer-to-Layer Mixing Chain

This is the key structural idea:

$$x \to H \to \text{Layer1} \to H \to \text{Layer2} \to H \to \cdots$$

Each step:

- re-mixes everything
- feeds a new global view to the next layer

So:

> The output mix of one layer becomes the input mix of the next

You don’t need to track coordinates—just think:

- “everything is already mixed”

---

8. Case 3: Output Layer (Bias Matters Again)

At the very end:

- you apply FWHT again

But now you're going back to:

- a structured output space

So again, you should apply:

$$(\text{final output}) = D_{out} H x^{(L)}$$

where:

- D_{out}: fixed random ±1 pattern

Why?

Same reason as input:

- avoid bias toward specific Walsh components
- ensure fair reconstruction into output space
- Backpropagation will see $HD_{out}e$, where e is the error vector.

---

9. Backpropagation Is Easy

FWHT is (up to scaling) self-inverse:

$$H^{-1} = H$$

So:

> Backprop through FWHT = apply FWHT again

No complications:

- no learned parameters
- no numerical instability
- very efficient

---

10. Implementation Summary

Input

1. Apply fixed random sign flips
2. Apply FWHT

Network core

- Alternate:
    - FWHT
    - small / sparse / nonlinear layers
- No need for sign flips here

Output

1. Apply FWHT
2. Apply fixed random sign flips

---

11. Big Picture

- FWHT = **global mixing operator**
- Sign flips = **bias correction for structured data**

Together:

> You turn local, simple computations into a globally coordinated system without needing deep stacks of layers.

---

12. One-Line Intuition

> Randomize$_1$ → Mix globally → Compute locally → Mix again → Randomize$_2$